The invention relates to a computer system providing an object-based virtual machine environment, in which middleware runs successive applications on a single virtual machine. The computer system noted includes storage for storing objects for running the aforementioned successive applications. The storage is logically divided into three heaps. A "heap" is a block of memory that belong to a program but have not yet been given a specific use.

The heaps comprise:

a system heap which is not garbage collected;
a middleware heap which is garbage collected; and
a transient heap which is cleared in between successive applications.

Garbage collection refers to clearing out objects that are taking up space in memory but are no longer in use by the, e.g., JAVA program.

In this approach, at the end of an application, the virtual machine stays up to execute the next application, rather than creating a virtual machine within a process to load and run an application, and then tearing down the entire process and runtime at the end of the application. Not creating the virtual machine environment per application increases the volume and throughput of applications a system can manage.

The present invention uses multiple heaps to retain persistent data and transient data. This use of multiple heaps enables a single virtual machine to be easily resettable, thereby avoiding the need to terminate and start a new virtual machine for each application. The use of multiple heaps also enables a single virtual machine to retain data and objects across multiple applications, thus avoiding the computing resource overhead of relinking, reloading, reverifying, and recompiling classes that have already been used by previous applications.

In the preferred embodiment, the memory hierarchy also includes a system heap where classes are loaded, linked, verified, initialized and compiled. Subsequent applications can reuse the classes in the system heap and need not go through the overhead of reloading, linking, verifying and compiling them again. Middleware can create its persistent or resettable objects in the middleware heap. Any necessary garbage collection is preferably performed in between applications, thereby avoiding this overhead during the lifetime of an application, and so improving client response times.

Applicants emphasize that the application data that are used only during the lifetime of an application are created in the transient heap, which is cleared after every application.

As noted above, in accordance with the present invention, a virtual machine has three types of heaps; a system heap, a middleware heap and a private (or transient) heap.

Classes are loaded and linked into the system heap, which will not be cleared or reset. Therefore class objects once loaded, linked and compiled into the system heap need not be reloaded, relinked or recompiled for subsequent applications.

The middleware heap exists for middleware applications to store data that could be reset or must persist for subsequent applications.

The private heap is used for application specific data and will always be cleared after an application runs and before the next application is scheduled into the virtual machine (hence it is generally referred to herein as the transient heap). This definition of "transient heap" is present in Claim 1.

Accordingly, the present invention provides a computer system providing an object-based virtual machine environment, in which middleware runs successive applications on a single virtual machine, said system including storage for storing objects for running said applications, said storage being logically divided into the three heaps listed above.

In this approach, at the end of an application, the virtual machine stays up to execute the next application, rather than creating a virtual machine within a process to load and run an application, and then tearing down the entire process and runtime at the end of the application. This leads to saving the cost of process tear down and start up which is expensive. Having a long running, reusable virtual machine also helps save the cost of class linking, loading and initialization which is also expensive. Furthermore, not creating the virtual machine environment per application increases the volume and throughput of applications a system can manage.

The preferred embodiment of the virtual machine environment of the present invention uses multiple heaps to retain persistent data and transient data. This use of multiple heaps enables a single virtual machine to be easily resetable, thereby avoiding the need to terminate and start a new virtual machine for each application. The use of multiple heaps also enables a single virtual machine to retain data and objects across multiple applications, thus avoiding the computing resource overhead of relinking, reloading, reverifying, and recompiling classes that have already been used by previous applications. In the preferred embodiment, the memory hierarchy also includes a system heap where classes are loaded, linked, verified, initialized and compiled. Subsequent applications can reuse the classes in the system heap and need not go through the overhead of reloading, linking, verifying and compiling them again. Middleware can create its persistent or resettable objects in the middleware heap. Any necessary garbage collection is preferably performed in between applications, thereby avoiding this overhead during the lifetime of an application, and so improving client response times. Application data that are used only during the lifetime of an application are created in the transient heap, which is cleared after every application.

In the preferred embodiment, a card table is used which is marked whenever a possible reference to the transient heap is created. This enables a potentially faster route to allowing the virtual machine to reset the transient heap, compared to a full garbage collection, thereby increasing throughput of the virtual machine.

Thus to summarize, the preferred embodiment has the virtual machine has the system heap, a middleware heap and a private heap. Classes are loaded and linked into the system heap, which will not be cleared or reset. Therefore class objects once loaded, linked and compiled into the system heap need not be reloaded, relinked or recompiled for subsequent applications. The middleware heap exists for middleware applications to store data that could be reset or must persist for subsequent applications. The private heap is used for application specific data and will always be cleared after an application runs and before the next application is scheduled into the virtual machine (hence it is generally referred to herein as the transient heap).

With this background in mind, Applicants respectfully submit that the references cited are not pertinent with respect to the present invention.

The Examiner is respectfully requested to reconsider the rejection of Claims 1 - 7, 15 and 16 under 35 U.S.C. §103(a) as being unpatentable over in view of U.S. Patent 6,275,985 to Ungar in view of *"Concurrent Compacting Garbage Collection of a Persistent Heap* (hereinafter "O'Toole").

The Ungar system facilitates developing an application that implements garbage collection using a first compiler and proxy objects and then compiling the application with a second compiler that provides support for efficient garbage collection. During execution of the application, pointers within the system stack point indirectly to data objects through the proxy objects. The purpose of the Ungar system is to implement garbage collection in his application. Based upon a detailed reading of the Ungar

reference, Applicants respectfully emphasize that Ungar unequivocally teaches the use of a system heap and a transient heap for efficient means of garbage collection.

Those skilled in the art understand that In certain programming languages including C and Pascal, a "heap" is an area of pre-reserved computer main storage (memory) that a program process can use to store data in some variable amount that won't be known until the program is running. For example, a program may accept different amounts of input from one or more users for processing and then do the processing on all the input data at once. Having a certain amount of heap storage already obtained from the operating system makes it easier for the process to manage storage and is generally faster than asking the operating system for storage every time it's needed. The process manages its allocated heap by requesting a "chunk" of the heap (called a *heap block*) when needed, returning the blocks when no longer needed, and doing occasional "garbage collecting," which makes blocks available that are no longer being used and also reorganizes the available space in the heap so that it isn't being wasted in small unused pieces.

The term "heap" was apparently inspired by another term, "stack." A stack is similar to a heap except that the blocks are taken out of storage in a certain order and returned in the same way. In Pascal for example, a *subheap* is a portion of a heap that is treated like a stack. It is imperative to keep in mind in reading the Ungar reference that Ungar is using both a heap and a stack in the invention and they are not identical. The skilled artisan would not confuse the stack with the heap. Thus the specific function of these two elements as used in the invention make it questionable as to whether the cited portions of the Ungar disclosure do in fact read on the elements of Applicants' claims as the Examiner asserts that they do.

The Examiner states on page 8 of the Official Action: "...a transient heap which is cleared in between successive applications and which has NO garbage collected within the lifetime of the application" (Emphasis added), and cites Column 5, lines 1 - 45 in support

of this assertion. Column 5, line 30 etc. states: "System heap 312...contains proxy objects to help in the garbage collection process..." The fact is that Applicants do not collect garbage in either the system heap or the transient heap and the cited excerpt establishes that Ungar does collect garbage in the system heap and the transient heap.

Applicants have considered the specific disclosure found in the Ungar reference with respect to the manner in which the system operates. This is essential as it is the primary reference and one needs to consider the propriety of the combination of it with the O'Toole reference, i.e., is there a suggestion within the teaching that warrants the combination.

O'Toole uses a heap to safely store transactional objects. O'Toole does not describe a middleware heap in his disclosure. O'Toole is in reality a means for replicating garbage collection for a persistent heap. The O'Toole disclosure teaches how a persistent heap is created. Essentially, Ungar and O'Toole teach new ways to collect garbage collect.

Applicants' invention differs from the combination of Ungar and O'Toole in that in Applicants' invention, there is a middleware heap which is garbage collected; further, there is a system heap and a transient heap, neither of which is garbage collected.

With respect to the Official Action's Claim Rejections found in paragraphs 3 to 9 starting at page 5 Applicants submit the following:

1) There is a substantial difference between Unger's system and the present invention in that the present invention does not require

- Proxy objects
- Two Compiler passes to (1) create proxy objects and then (2) Create stack maps

Unger's Patent in Figure 7 requires traversing the proxy area (left column below) in the first compile pass and in the 2nd compile pass, creating a stack map and traversing each frame in the stack map (right column below).

"....."

LOOK IN PROXY AREA FROM START TO FREE POINTER, IF WORD POINTS INSIDE APPLICATION HEAP AND IT IS NOT ON FREE LIST, RETURN ADDRESS OF WORD	TRAVERSE STACK, FOR EACH FRAME TRAVERSE STACK MAP AND RETURN LOCATIONS OF POINTERS

FIG.7 ...."

The present invention does not require a proxy area or a stack map nor double compile passes. Pointers do not have to be checked nor frames traversed. Garbage collection for our transient heap involves zeroing the entire transient heap. No double compile passes are required.

The Ugar reference does not contain the teaching that the Examiner contends that it does. Thus the assertion that certain elements claimed by Applicants are disclosed in the Ungar reference is not accurate.

The Examiner is respectfully requested to reconsider the rejection of Claims 8 - 10 under 35 U.S.C. §103(a) as being unpatentable over in view of U.S. Patent 6,275,985 to Ungar and "Concurrent Campacting Garbage Collection of a Persistent Heap (hereinafter "O'Toole") in view of United States Patent 6,249,793 to Printezis, et al.

The arguments with respect to the Ungar and O'Toole references as set forth above are incorporated by reference at this site in response to the rejection including the Printezis, et al. reference.

There is no proper basis to combine the references Ungar, O'Toole and Printezis references as Ungar implements garbage collection in his system heap and O'Toole has no middleware heap. These features are contrary to Applicants' invention. Thus the defects present in these two inventions cannot rehabilitate these references when combined with Printezis, et al. to render the present invention obvious. The only element found in Printezis et al. which is relevant to the present invention is its disclosure of a card table-*"data structure."*

With respect to the Official Action's Claim Rejections found in paragraphs 10 to 13 starting at page 9 Applicants submit the following:

A significant distinction between Printezeis (6,249,793) and the present invention is that the instant application does not require:

- Reserve Copy locations
- Write Barriers
- That the program be suspended during garbage collection

The features mentioned above are depicted in Figure 9 of Printezis and are required for it to operate.

The Examiner is respectfully requested to reconsider the rejection of Claims 11 - 14 under 35 U.S.C. §103(a) as being unpatentable over in view of U.S. Patent 6,275,985 to Ungar and "Concurrent Compacting Garbage Collection of a Persistent Heap (hereinafter "O'Toole") in view of United States Patent 5,950,008 to van Hoff.

The arguments with respect to the Ungar and O'Toole references as set forth above are incorporated by reference at this site in response to the rejection including the van Hoff reference.

The van Hoff reference discloses a program interpreter for interpreting object oriented programs in a computer system having a memory that stores a plurality of objects and a plurality of methods. The van Hoff reference uses class loaders to describe in which heap to create the objects. Applicants' class loader as disclosed, is used to define where to place the storage objects—i.e., where the storage objects should be created.

With respect to the Official Action's claim rejections found on pages 14 to 18 on pages 11 – 12, a substantial difference between Van Hoff's patent (5,950,008) and the instant invention is that the Van Hoff invention describes the use of class loaders to fetch object classes (Figures 5,6 in the reference patent) and the instant invention is concerned with serially reusing the same JVM. Applicants' invention does not claim a new method of creating class loaders.

In each of the 35 U.S.C. 103(a) rejections, the Examiner in the Official Action has misinterpreted the disclosures and selected concepts improperly from the references out of context as the basis for the rejections. The rejections are a piecemeal construction of the invention. Such piecemeal reconstruction of a prior art patent in light of the instant disclosure is contrary to the requirements of 35 U.S.C. 103 as set forth specifically below. Ungar and O'Toole alone, or in combination with Printezis or van Hoff, do not disclose or even imply the system and/or the method of the present invention. In the rejection, the Examiner is picking and choosing elements to the exclusion of what the references as a whole teach to one skilled in the art. To arrive at Applicants' invention, the person skilled in the art would have to randomly pick and choose among a substantial number of different elements and/or systems found in the references with absolutely no guidance whatsoever to direct him/her to the system and method claimed by Applicants. Based upon the skilled artisans knowledge of the properties of the systems disclosed in the prior references cited and their respective objectives and how they are implemented, it is unlikely that the Ungar reference would be used in combination with O'Toole and Printezis or van Hoff.

In order to analyze the propriety of the Examiner's rejections in this case, a review of the pertinent applicable law relating to 35 U.S.C. § 103 is warranted. The Examiner has applied the various references discussed above using selective combinations to render obvious the invention.

The Court of Appeals for the Federal Circuit has set guidelines governing such application of references. These guidelines are, as stated are found in Interconnect Planning Corp. v. Feil, 774 F.2d 1132, 1143, 227 USPQ, 543, 551.

When prior art references require selective combination by the court to render obvious a subsequent invention, there must be some reason for the combination other than hindsight gleaned from the invention itself.

A representative case relying upon this rule of law is Uniroyal, Inc. v. Rudkin-Wiley Corp., 837 F.2d 1044, 5 USPQ 2d 1434 (Fed. Cir. 1988). The district court in Uniroyal found that a combination of various features from a plurality of prior art references suggested the claimed invention of the patent in suit. The Federal Circuit in its decision found that the district court did not show, however, that there was any teaching or suggestion in any of the references, or in the prior art as a whole, that would lead one with ordinary skill in the art to make the combination.

The Federal Circuit opined:

> *Something in the prior art as a whole must suggest the desirability, and thus the obviousness, of making the combination.* [837 F.2d at 1051, 5 USPQ 2d at 1438, citing Lindemann, 730 F.2d 1452, 221 USPQ 481, 488 (Fed. Cir. 1984).]

Applicants respectfully submit that there is no basis for the combination of the aforementioned references cited by the Examiner. Applicants have pointed out how the references teach in different directions. The Examiner has selected elements and steps from the cited references for the sake of showing the individual elements and/or steps claimed without regard to the total teaching of the references.

The Examiner in his application of the cited references is improperly picking and choosing. The rejection is a piecemeal construction of the invention. Such piecemeal reconstruction of the prior art patents in light of the instant disclosure is contrary to the

requirements of 35 U.S.C. § 103.

> *The ever present question in cases within the ambit of 35 U.S.C. § 103 is whether the subject matter as a whole would have been obvious to one of ordinary skill in the art following the teachings of the prior art at the time the invention was made. It is impermissible within the framework of Section 103 to pick and choose from any one reference only so much of it as will support a given position, to the exclusion of other parts necessary to the full appreciation of what such reference fairly suggests to one of ordinary skill in the art. (Emphasis in original) In re Wesslau 147 U.S.P.Q. 391, 393 (CCPA 1965)*

This holding succinctly summarizes the Examiner's application of references in this case, because the Examiner did in fact pick and choose so much of each of the references cited to support his position and did not cover completely in the Office Action the full scope of what these varied disclosure references fairly suggest to one skilled in the art.

Further, the Federal Circuit has stated that the Patent Office bears the burden of establishing obviousness. It held this burden can only be satisfied by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the reference.

Obviousness is tested by "what the combined teachings of the references would have suggested to those of ordinary skill in the art." In re Keller, 642 F.2d 413, 425, 208 USPQ 871, 881 (CCPA 1981). But it *"cannot be established by combining the teachings of the prior art to produce the claimed invention, absent some teaching or suggestion supporting the combination."* ACS Hosp. Sys., 732 F.2d at 1577, 221 USPQ at 933. [837 F.2d at 1075, 5 USPQ 2d at 1599.

The court concluded its discussion of this issue by stating that teachings or references can be combined only if there is some suggestion or incentive to do so.

In the present case, the skilled artisan, viewing any or all of the references would be directed toward a totally different system than is called for in the present invention. The system emanating from the Ungar and O'Toole references would be a system that uses that has no middleware heap with NO garbage collection in the system heap and the transient heap. The references are teaching in opposite, or at least inconsistent directions. There is no proper basis to combine them.

Applicants have attempted in this response to amend the claims and to place these amended claims in a form which should result in their allowability.

If the Examiner wishes to discuss via telephone the substance of any of the proposed claims contained herein with the intent of putting them into an allowable form, Applicants' attorney will be glad to speak with him at a mutually agreeable time and will cooperate in any way possible.

Any fees which result from the claims added herein should be charged to Deposit Account 50-0510.

In view of the arguments and modifications to the claims, allowance of this case is warranted. Such favorable action is respectfully solicited.

Please address all future correspondence in this application to the undersigned at the address listed below.

Please grant a one month extension of time within which to respond to the Official Action noted above. The Commissioner is authorized to charge the $120.00 extension fee to Deposit Account 02-1651.

Respectfully Submitted,

Thomas A. Beck
15 Alameda Place
Mount Vernon, NY 10552
(860) 921-1358

I certify that this amendment is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to: *Assistant Commissioner of Patents, P.O. Box 1450. Alexandria, VA 22313-1450.*

Signature _____ Date: March 2, 2007

Name: Thomas A. Beck